RAG with LLamaIndex

In this report, I will detail the process and results of building a Retrieval-Augmented Generation (RAG) system using the Llama-Index framework. The aim was to develop a comprehensive system capable of efficiently processing and retrieving data, enhancing the capabilities of a language model. This project followed a structured approach, encompassing three major parts: Data Processing, Basic RAG, and Advanced RAG. The Advanced RAG section includes Backoff with a custom query engine, different chunking techniques, HYDE, re-rank, multi-step query, and router query engine. This structure ensured a systematic and thorough approach to developing the RAG system.

For testing, please run all the cells from beginning, then run the Test part.

Setup

The project began by setting up the Language Model and Embedding model using OpenAI's GPT-3.5-turbo for language processing and text-embedding-3-small for embeddings.

Data Processing

Data Extraction

The initial step involved scraping data from the Llama-Index blog (https://www.llamaindex.ai/blog). The data extraction process was automated using web scraping tools, ensuring the retrieval of relevant articles and content.

Data Cleaning and Normalization

Post extraction, the data underwent a thorough cleaning process. This involved removing HTML tags, special characters, and redundant spaces, and normalizing text formats to ensure consistency. The data was then structured to facilitate efficient processing and retrieval.

Building the Basic RAG System

Chunking

Chunking the data was performed using the SentenceSplitter from the Llama-Index core node parser. The SentenceSplitter was configured with a chunk size of 1024 tokens and an overlap of 20 tokens to ensure optimal chunk sizes for processing.

Query with ChromaDb (Persistent Index)

The ChromaDB client was set up to create a persistent database for storing and retrieving vectors. The database was configured to delete any existing collections and create a new collection for storing vectors. The vector store was then set up using the ChromaVectorStore, and a storage context was created from these defaults. This storage context was used to set up a VectorStoreIndex for querying the stored vectors. Queries were processed using different response modes to evaluate the effectiveness of ChromaDB's indexing and retrieval mechanisms.

Multiple Indices with Persistent Docstore

In addition to ChromaDB, multiple indices were created using a persistent document store to compare retrieval results. It was observed that ChromaDB's storage and retrieval mechanisms differed from the native Llama-Index, resulting in less comprehensive answers when using ChromaDB. The configuration ensured that the system displayed the specific sources of data used to answer queries, including the blog titles and paragraph positions of the information.

Custom Display

A custom function was implemented to display the sources used for answers, including specific blog titles and relevant paragraphs.

Enhancements to the RAG System

Backoff Mechanism

A backoff mechanism was applied to handle rate limits for the LLM (gpt-3.5-turbo). This prevented the chatbot from encountering errors (error 429) due to exceeding rate limits.This backoff mechanism was implemented using RAGStringQueryEngine, a custome query engine.

Different Chunking Techniques

In addition to the SentenceSplitter, two other chunking techniques were tested to optimize data retrieval:

- **HierarchicalNodeParser**: This parser chunks input data into hierarchical layers, with each node linked to its parent. When used with the AutoMergingRetriever, it replaces retrieved nodes with their parent nodes when a majority of child nodes are retrieved. This ensures the LLM has a more complete context for response synthesis, enhancing accuracy and relevance.
- **TokenTextSplitter**: Observations indicated that a chunk size of 1024 tokens yielded better results compared to 2024 tokens.

Re-Ranker Integration

The Llama-Index LLM rerank approach was used to enhance retrieval accuracy. This approach involves using the LLM to decide the relevance of documents or text chunks. The input prompt consists of candidate documents, and the LLM selects the relevant documents and scores their relevance with an internal metric. The integration of a re-ranker model prioritized the most relevant documents. Nevertheless, in this case, no noticeable difference in the result was found.

Hypothetical Document Embeddings (HYDE)

Hypothetical Document Embeddings (HYDE) were implemented to provide more contextually accurate and relevant responses. This technique refines the retrieved data before passing it to the LLM. This seems to significantly improved the comprehensiveness and detail of responses.

Multi-Step Query

Multi-step querying was employed to handle complex queries by breaking them down into simpler sub-queries. This method allowed the system to provide more precise answers. The MultiStepQueryEngine was used to execute these multi-step queries, ensuring a more accurate and thorough response.

However, the queries generated misinterpreted the original questions.This suggest the need to fine tune the MultiStepQueryEngine offered by Llama index for more accurate result.

Router Query Engine

A router query engine was used to manage the selection and routing of queries to the most appropriate indices or data sources. This included both single and multi-selector configurations to optimize query processing. The QueryEngineTool and ToolMetadata from Llama-Index were used to set up these tools. No noticeable difference between the results of single and multi-selector was found.

Observations and Insights

1. **HierarchicalNodeParser**: This parser was used in conjunction with the AutoMergingRetriever to chunk data into hierarchical layers, facilitating better context synthesis for the LLM.
2. **TokenTextSplitter**: A chunk size of 1024 tokens provided better results compared to 2024 tokens.
3. **SentenceSplitter**: Chunk size did not significantly affect the results.
4. **Response Modes**: Different response modes yielded varying results, with tree summarize being the most effective for certain queries.
5. **ChromaDB Indexing**: Indexing and retrieval behavior varied with ChromaDB, highlighting its distinct mechanisms.

Conclusion

This project developed and enhanced a RAG system using the Llama-Index framework. Through systematic data collection, preprocessing, system building, and performance improvements, the system demonstrated its capability to efficiently retrieve and process information. The insights gained from different chunking techniques, indexing methods, and response modes provide valuable guidance for further optimizations.